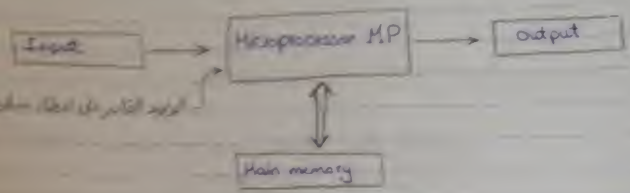
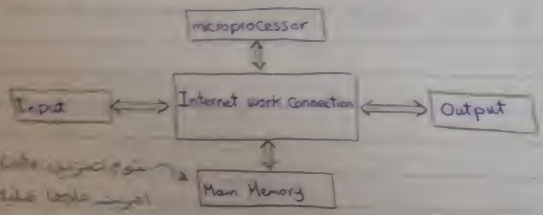
Ch 1 Microprocessor (Introduction)





Control processing Unit (cpu) ≡ Machine

→ Cpu Components

1. ALU
2. CU (Control Unit)
3. Registers level 1 Cache

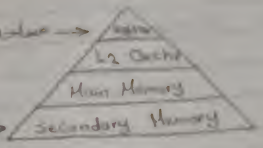
تتكلم في وظائف الجهاز

→ Cpu functions:

1- Execode arithmatic & logic instruction
(Fetch - decode - execode)

→ Manage the program Flow (make decisions)
* Zero * negative * overFlow ...

مساحة أقل ، أسرع ، أغلى سعر ←



أكبر مساحة ، أقل سرعة ، أقل سعر ←

→ Transfer Data between itself and memory

processor

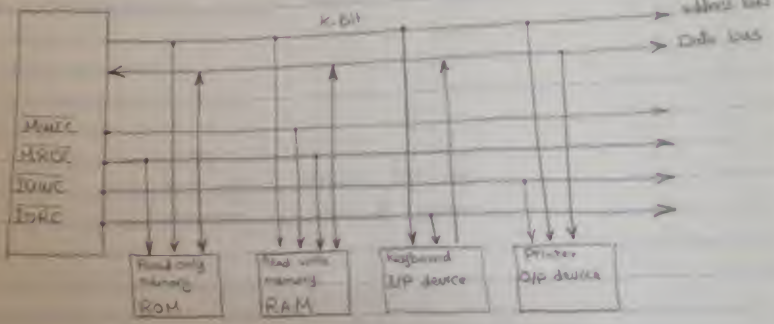
Multi-processor ← أشكال → Multi-Core

Bus: group of common wires connect between parts of device.

1. **Address Bus:** →
   K-bit ⟶ 2^K devices.
2. **Data Bus:** ↔
   devices ↔ MP ← كلما زاد كلما زاد معدل نقل البيانات من وإلى الـ MP
3. **Control Bus:** ↔
   devices ↔ MP ← ينقل Control Signal
   كلما زاد كلما كان أفضل

←

Draw the block diagram of a computer System Showing the address data and Control bus Signal ?



+ Simple Control Signal

MWIc — memory write Control Signal
MRDC — memory read Control Signal
IOWC — I/O write Control Signal
IORC — I/O read Control Signal

→ MP → device التفاعل بين

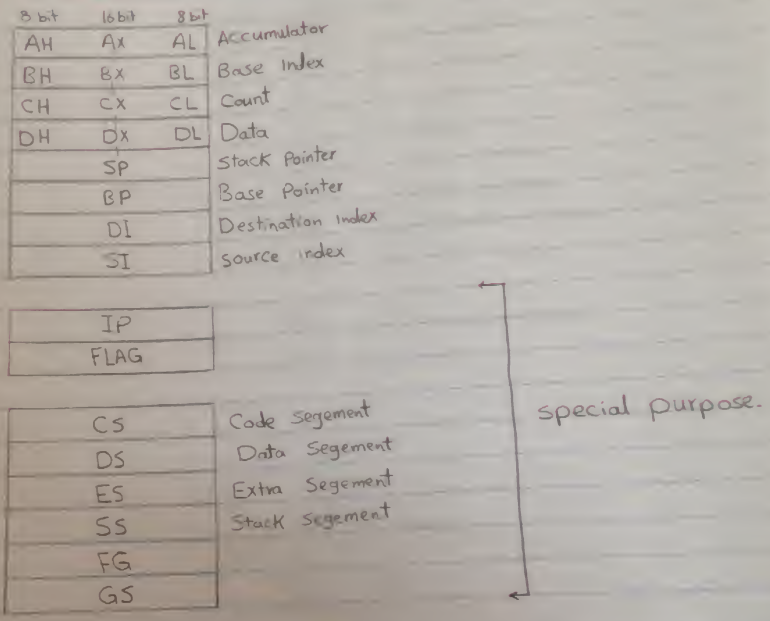
1- MP ⇝ Send address
2- Control Signal MRDC
3- Data transfer.

The memory system is divided into 3 main parts:

1. Transient program area (TPA)
2. System Area
3. Extended memory system (EMS)

1.
- → holds
- → The length of TPA is 6410 KB
- → programs → interrupt vector

Ch 2 : The Microprocessor & Its Architecture



Programming Model Of The Intel 8086 Pentium 4

32 bit		8bit	16bit	8bit	
EAX		AH	AX	AL	Accumulator
EBX		BH	BX	BL	Base Index
ECX		CH	CX	CL	Count
EDX		DH	DX	DL	Data
ESP			SP		Stack Pointer
EBP			BP		Base Pointer
EDI			DI		Destination Index
ESI			SI		Source Index

	IP	Instruction Pointer
	FLAGS	Flags

CS	Code
DS	Data
ES	Extra
SS	Stack
FS	
GS	

The Programming Model Of The Intel 80386 Pentium 4

كل 4 bit في الشاشة بتطبع 1 في السادس عشر ←

20 bit ⟶ 1XXXXX1 Hexa.

Real Mode

Segment	Off Set	Special Purpose
CS	IP - EIP	Instruction address
SS	SP - BP ESP / EBP	Stack address
DS	BX DI SI 8bit 3 number 16bit	Data address
ES	DI EDI for string Instructions	String destination address
FS - GS	No default	General address

Programming Model

program Visible
- They are used during app-prog
- ex: EAX AX AH, AL EBX ...
→ Mov Ax, Bx

program invisible
- not addressed directly during app-prog
- only 80286 - P4 Contain this type
- ex: FLAG EFLAG

SUB Ax Bx بعد تنفيذ أمر

Purpose of use

General Purpose
EAX , EBX
Hold Data address or off set +
البعد عن البدايه

Special purpose
CS. DS ES. SS

Int X = 3
address ↑ data ↑

Addressing modes (How the memory is accessed)

Addressing mode ← كثيرة تكثر طرق تنظيم البيانات داخل الذاكره
وظيفتها الوصول الى الذاكره

1- Real mode 1 MB only
2- Protected mode upto 4GB

« Real mode »

→ Accumulator EAX AX . AL . AH
Can be used as 32bit → EAX , 16 → AX , 8bit → AH, AL

II Special purpose .
طرق أساسى فى عمليات الضرب والقسمة ←

→ Div AX , BX : $\frac{AX}{BX}$

AX : يوضع به الجزء الصحيح
DX : (Data register) يوضع به باقى القسمه "الكسر"

→ Mux AX , BX
16 * 16 → 32 bit

→ Mux EAX , EBX
32 , 32

DX high bytes | AX low bytes

EDX high bytes | EAX low bytes

Base Index

Can be used as 32 bit → EBX . 16 bit → BX
8 bit → BH . BL

General purpose → hold offset For Data segement

Note : In Real Mode

We have 64 KB processor . 5 bit memory

→ To access the memory الوصول للذاكرة

← processor الى

$64K = 2^6 \cdot 2^{10} = 12^{16}$

$(16\ bit)_8 \rightarrow (4\ bit)_{Hexa}$.

↝ From 4 bit processor how to reach 5 bit memory
we add zero From the left side to Base

→ to reach to any address in the memory we need Base and offset

الـ Base is in Code segement

∴ Current address$_{(real)}$ = $(CS)_0$ + offset

Ex: CS = 4000 Hex . IP = 2342

Solution

Current address = $(CS)_0$ + IP = 40000 + 2342 = 42342 Hex

Count

To Count in any process with loop we need big register

loop CX . ECX

Repetition CX

Shift CL we need small number of loops

Data

Can be used as 32 bit → EDX 16 bit → DX

8 bit → DH , DL

General purpose → Old , Mux العمليات والقسمة

Stack pointer

Can be used as 16 bit → SP

hold the off set of stack segement (SS) ← الأولوية له

Base pointer

Can be used as 16 bit → BP

used as off set for stack segement (ss)

Destination Index

Can be used as 16 bit → DI له الأولوية بعد BX

hold the off set of Data segement

Source Index

Can be used as 16 bit → SI له الأولوية بعد DI

hold the off set of Data segement

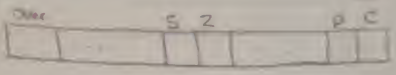
Instruction Pointer

Can be used as 16 bit → IP

hold the off set of current instruct to be excuted (in Code segement)

Ch 4 : Data Movement Instructions

Flag - Invisible



Can be used as 16 bit - 32 bit

→ Help microprocessor to take the correct way

1. Over Flow happened when → +VE + +VE = -VE
   → -VE + -VE = +VE

2. Carry (C)

Carry 1111 0110 +
(1) ← 1111 0000

3. Parity (P)

1 → number of one's is even

0 → number of one's is odd

تستفيد من هذا عندما يتم نقل Data من مكان لآخر نتأكد من أن الـ Data تم نقلها بدون فقد صح اذا كان عدد الـ 1 زوجي بعد النقل وكذلك الـ 0 كان النقل تم بالصوره المطلوبه والعكس يكون به فقد في Data

Ex: SS = 3245 H

BP = 2342

Sol

Base address = 33450 H

Current address = 33450 + 2342 = 35792

Ending address = Starting + FFFF

= 33450 + FFFF

Used in Two dimension ...

Ch 3 Addressing modes (Real Mode)

1) Immediate Addressing Mode

I → R

Examples:
- Mov BL, 44H
- Mov BH, 44H
- Mov CL, 10011100 B
- Mov ECX, FC23F34EH

2) Register Addressing Mode

R ↔ R

- Mov CL, BL
- Mov CX, BX
- Mov EAX, EBX
- Mov DS, CX
- Mov DS, CS

Mov CS, BX (X)

Mov BX, CS (✓)

3) Direct Addressing Mode

R ↔ M (memory)

- Direct addressing
  - R: EAX, AX, AH, AL
- Displacement (offset) addressing
  - Any Register

Direct:
- Mov AL, Num
- Mov AX, Num
- Mov Num, AX
- Mov EAX, Num

Displacement:
- Mov CL, Num
- Mov CX, Num
- Mov EBX, Num
- Mov DX, [1023]

4) Register Indirect Addressing Mode [ ]

(BP, BX, SI, DI)

R ↔ [M]

- Mov AX, [BX]
- Mov [BP], DL
- Mov EDX, [DI]
- Mov [BX], [SP] (X)

5) Base_Plus_Index Addressable Mode

BX → BX
EBP → BP
ESI → SI
EDI → DI

use with one dimention array

R ↔ M

- Mov DX, [BX+DI]
- Mov WORD / DWORD PTR [BX+DI], DX

6) Register_Relative Addressable Mode

BP, BX, SI, DI
EBP, EBX, ESI, EDI

offset

R ↔ M

- Mov AX, [BX+1000]
- Mov AX, 1000 [DI]
- Mov WORD / DWORD PTR [BX+1000], AX

7) Base_relative plus Index Addressable Mode

Used in Two dimention Array

R ↔ M

- Mov WORD PTR 1000 [BX+DI], A
- Mov DH, [BX+DI+20H]

Effective Address (EA)

No EA

Notes

( )$_{10}$ عشري

( )$_{B}$ ثنائي

( )$_{H}$ سادس عشر

R_{Source} (R_S)

Register الذي سيحصل منه على البيانات

R ↔ R has the same size

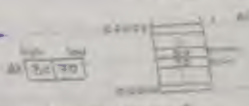
لا يجوز بأي حال من الأحوال تغيير Code segment لأنه يحتوي على عنوان أول أمر داخل الذاكرة وكذلك لا يجوز النقل من segement الى آخر

E.A = memory = Num

1→ Memory ~ Byte addressable

Byte by Byte مترجمة

2→



little endian assignment

$EA = (seg)_{+0} + offset$

① $EA = (DS)_0 + BX$

② $EA = (SS)_0 + BP$ ③ $EA = (DS)_0 + DI$

عند نقل بيانات من Memory الى Register لا بد من تحديد المساحة المطلوب تحويل البيانات لها حيث أن Memory byte addressable

Mov [BX], AL ← لم يتم تحديد مساحة التحويل

لم يتم التحويل الى

Mov Byte PTR [BX], AL

Mov WORD PTR [BX], AL

Mov DWORD PTR [BX], AL

$EA = (DS)_0 + BX + DI$

Starting address

ويظل effective address ثابت

$EA = (DS)_0 + BX + 1000$

عند نقل بيانات من Memory الى Register لا تحدد المساحة للتخزين فلا توجد مشاكل

$EA = (DS)_0 + \begin{matrix} BX \\ DI \\ 1000 \end{matrix}$

Ch 4 : Data Movement Instructions

1- Assembler & Decoassembler

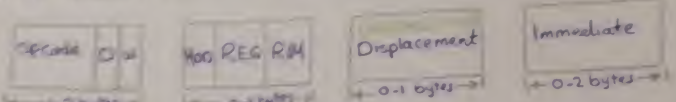
2- Stack (push, pop)

3- Instructions

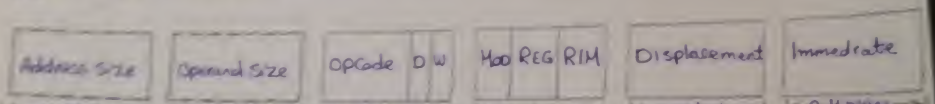
Assembler & Decoassembler

Mode of Instruction

1 16 bit instruction mode (8086)



2 32 bit instruction mode (80386 above)



To Change From 16 bit to 32 bit Follow the steps

1

Operand Size (66 H)	Address Size (67 H)	mode
X	X	- 16-bit instruction mode with its default (8 - 16 bit) registers
✓	X	- 16 bit instruction but works with (8 - 32 bit) registers.
X	✓	- 32 bit instruction works with its default (8 - 32 bit) register
✓	✓	- 32 bit instruction but works with (8 - 16 bit) registers.

[2] | Opcode | D | W |

→ Opcode: has machine code of the instruction (Mov, Sto, Jump)
we get the code from the table
Ex: Mov → 100010

→ D → D=1 (when we transfer from Memory to register)
→ D=0 (when we transfer from register to Memory)

→ W → W=1 (Word / Dword)
→ W=0 (Byte)

[3] | MOD | REG | R/M |

→ MOD: its machine code can be get from this table

MOD	Function		Ex:
00	No Displacement	R/M is memory	Mov AL, [DI]
01	8-bit sign-extended displacement		Mov AL, [DI+2]
10	16-bit displacement		Mov AL, [DI + 1000₁₀]
11	R/M is a register		

→ REG: its machine code can be get from this table

Code	W=0 (Byte)	W=1 (Word)	W=1 (Dword)
0 0 0	AL	AX	EAX
0 0 1	CL	CX	ECX
0 1 0	DL	DX	EDX
0 1 1	BL	BX	EBX
1 0 0	AH	SP	ESP
1 0 1	CH	BP	EBP
1 1 0	DH	SI	ESI
1 1 1	BH	DI	EDI

Code	Seg
0 0 0	ES
0 0 1	CS
0 1 0	SS
0 1 1	DS
1 0 0	FS
1 0 1	GS

→ R/M → to get the code of R/M → all first we want to know if R/M = Memory or Register that depend on MOD

IF MOD = 11 → R/M = registers
we can get its code from register table

Code	W=0 (8bit)	W=1 (16bit)	W=1 (32bit)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

IF MOD = (00, 01, 10) → R/M is Memory
we can get its code
From this table

R/M code	Addressing mode	Function
000	DS:[BX+SI]	DS:[EAX]
001	DS:[BX+DI]	DS:[ECX]
010	SS:[BP+SI]	DS:[EDX]
011	SS:[BP+DI]	DS:[EBX]
100	DS:[SI]	Uses scaled-index byte
101	DS:[DI]	SS:[EBP]*
110	SS:[BP]* special addressing mode	DS:[ESI]
111	DS:[BP]	DS:[EDI]
	16-bit R/M memory addressing mode	32-bit addressing mode selected by R/M

4 Displacement

① 1 byte [ ] → Put as it
ex: 1 → 01 , 21 → 21

② 2 byte [ ][ ] → Swap high, low
ex: 1234 → 3412
ex: 123 → 2301

Stack
(pop & push)

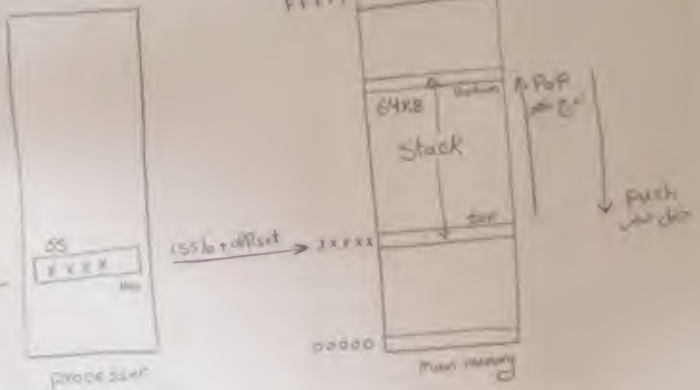
→ The PUSH and POP instructions are important instructions that store and retrieve data from the LIFO (last In First Out) stack memory

Stack (LIFO)

two operations

Pop — Push

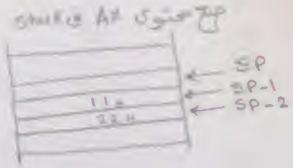
Pointer



Stack الخاص بالـ ← cyclic nature

→ Push AX (✓)

AX | 11H | 22H |
high byte — low byte



→ Push 5262H (✓)

→ Push [BX] (X) ← هنا لم يحدد المساحة التي يخزن بها البيانات في memory

EA = (DS)0 + BX

the Correct

Push Byte PTR [BX] (✓)

Push (word - Dword) PTR [BX] (✓)

→ Push CS (✓) → هنا لم يغير محتوى CS

* Hint

22H can be Put in memory as?

1 Byte | 22H |

2 word | 00 | 22 |

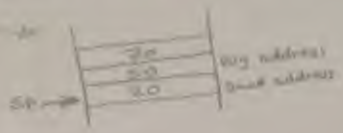
3 Dword | 00 | 00 | 22 |

Pop ← stack or data

Pop 2345 H (X)

Pop CS (X) ← we mustn't change CS

Pop BX (✓)



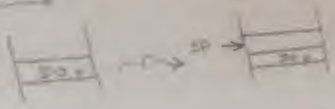
① BX | 50 | 20 |
high low

high address → high byte
low address → low byte

② Then SP = SP + 2

Pop Byte PTR [BX]

EA = (DS)₀ + BX



Instructions

- LEA
  - LDS, LSS, LFS, LGS, LES
    - * off set
- LOOS, StoS, Movs, INS, Outs.
  - * IN, Out
  - * X ChG
  - * BSwap
  - * CMoS

→ II LEA (load effective address)

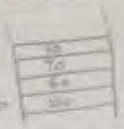
ex: LEA Ax, Numb ≡ Mov Ax, off set Numb

هذا الأمر معناه وضع بعد المكان Numb من البداية في Ax

ex: LEA BX, [DI] ⇝ EA = (DS)₀ + DI ≠ Mov BX, off set [DI] (X)

→ LDS

ex: LDS BX, [DI]



EA = (DS)$_0$ + DI = xxxx →

BX | 60 | 50

DS | 10 | 70

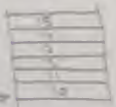
EA هيتم قدره بايجاد العنوان من
جهة داخل في memory
هيتم الامكان الي هنروح نوقع له

Hint

$b/t = 2^{byte}$

لو لقينا ان BX هيقدر يخزن
بايت واحد = 1 byte
فلازم نحجز في BX مكانين عشان يقدر يوقع العنوان
لان لو حجزنا مكان واحد بس هيكون 1 byte
فلما نحجز مكانين هيكون 2 byte
ثم نأخذ العنوان التاني ونضعه في
DS حيث LDS
الي الرقم

→ LSS



SX: LSS EAX, [BX+DI]

EA = (DS)$_0$ + BX + DI = xxxx →

EAX | 13 | 12 | 11 | 10

SS | 15 | 14

→ LES

ex: LES BX, [DI]

EA = (DS)$_0$ + DI = xxxx →

BX | 60 | 50 , ES | 80 | 70

→ LFS

ex: LFS BX, [DI]

EA = (DS)$_0$ + DI = xxxx →

BX | 60 | 50 , FS | 70 | 50

→ LGS

ex: LGS BX, [DI]

EA = (DS)$_0$ + DI = xxxx →

BX | 2 | 1 , GS | 4 | 3

LCS EAX, [DI] (X)

because CS mustn't be changed.

String Instructions

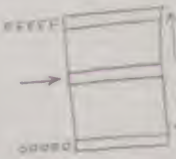
LODS, STOS, MOVS, INS, OUTS

This instructions depend on direction (D)

| 0 | FLAG

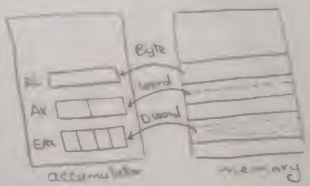
D=0 → Autoincrement mode
تسير في اتجاه زيادة العنوان

D=1 → Autodecrement mode
تسير في اتجاه نقص العنوان



1) LODS

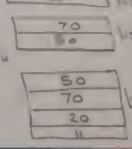
بتاخذ من memory وتضع في accumulator
ثم تزود او تقلل من DI



LODSB → byte
LODSW → word
LODSD → double word
المكان غير معروف مباشره يوصله EA عن طريق

LODS list → byte (DB)
LODS Data → word (DW)
LODS FROG → double word (DD)
المكان الرئيسي وضع بيه في الذاكره معروف مباشره

list: DB 20H
list2: DW 5070H
list3: DD 50702011H



Source → to → Destination

Data Segment — Extra Segment

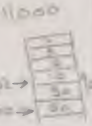
offset SI — DI

ex: DS = 1000, SI = 1002, D = 0
execute LODSW then LODSD

Solution

EA = DS + SI = 10000 + 1002 = 11002

SI new = SI old + 2 bit = 1002



AX | 30 | 50 |

To execute LODD

SI = 1002, DS = 1000, D = 0

EA = 10000 + 1002 = 11002

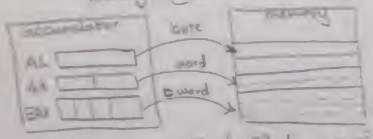
EAX | 7 | 6 | 5 | 0 |

SI new = SI old + 4 bit = 1002 + 4 = 1006

IF D=1 SI new = SI old − (1 → byte, 2 → word, 4 → dword)

2 STOS

يأخذ من memory و يضع في accumulator



C يزود أو يقلل حسب الـ offset

STOSB, STOSW, STOSD
ESTOS list (DB), STOS list2 (DW)
STOS list2 (DD)

3 MOVZX

زود باقي خانات Register بـ اصفار Zeros

MOVSX زود باقي المساحة (Memory) بنفس القيمة حسب الاشاره

4 XCHG تبادل المحتويات

→ Both places must have the same size
→ CS mustn't be Change

Examples:

XCHG AL, CL (✓)
XCHG CX, BP (✓)
XCHG Data, AX (✓)
XCHG DS, ES (✗)
XCHG CS, AX (✗)

5 BSWAP

→ Work only with 32 bit registers
i80486 → P3, P4)

EAX | 50 | 20 | 30 | 40 | before

EAX | 40 | 30 | 20 | 50 | after

6 CMOV (Conditional Move) نقل مشروط

CMOVZ → هذا الأمر معناه: يروح عالـ flag ع Z لو شافه، Z=0 لا يعمل شي، Z=1 يعمل

ex: CMOVZ AX, BX
IF Z=1 ∴ AX = BX
else Do nothing

→ CMOVC → Check Carry
IF C=1 → Move
else Do nothing

→ CMOVNC → Check No Carry
IF C=0 → Move
else Do nothing